Evolutionary Computing

[videos: 1) GOLEM project 2) Karl Sims creatures]

point: evolution is creative and simple: can we use it in AI?

evolution

emphasize it is <u>substrate-neutral</u>: "state space" mapping, operators, etc

evolution ∈ replication + variation + selection
- replication: DNA
- variation: mutation, recombination
- selection: fitness

issues:
- "survival of the fittest" tautology! fitness can be 1) number of offspring (tautology) 2) function of some properties (misleading: sometimes "weak" is successful, etc)
- "replicator" debate: Dawkins: DNA (selfish gene) or "meme"; Gould: many levels; group selection, etc
- complexity: complex organs (eye), complex systems (ecosystem): complexity increases

[evolutionary computing (EC): these issues do not exist more like breeding, not evolution]

notes:
- anyone can evolve that a) replicates b) has variation and c) has selection pressure (culture, etc) open source software, even 3D printers (RepRap)
- EC does not <u>model</u> evolution. it uses it as inspiration to solve problems (modeling is a very different task)

optimization - modeling - learning

- learning: finding <u>good</u> models of data from an explicit or implicit set of hypotheses
- modeling: fitting models to data in general
- optimization: selecting the <u>best</u> or <u>good</u> elements of a space (not necessarily models)

- evolutionary computing is for optimization
(thus for modelling and learning too)

- generic method: reasonable performance on wide set of problems, even when noise or implicit encoding of space elements, or where exact methods are expensive

- example where EC could but should not be used:
linear regression (many apps for learning and modelling)

x_1, y_1
$\vdots$
x_n, y_n

$$\min F(a,b) = \sum_i (y_i - (ax_i + b))^2$$

this can be solved exactly

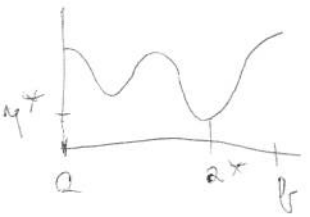
- examples where EC is a good idea: [wireless]; NP-hard problems, living in real environment, etc
(eg robots, etc)

evolutionary algorithm skeleton

- problem: given $f: A \to \mathbb{R}$ find min (or max) of f:

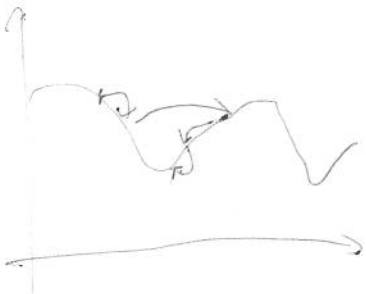
$$y^* = \min_{a \in A} f(a), \quad a^* = \arg\min_{a \in A} f(a)$$

for example: $A = [a, b]$



- stochastic hillclimber

```
1 x ← random from A
2 y = f(x)
3 repeat until "termination"
4     x' = small jump from x
5     y' = f(x')
6     if (y' < y) {x = x'; y = y'}
7 end-repeat
```



- EA is this idea, with some more tricks
- NOT a model of evolution!

-- the evo. loop

- use a population (= multiset over A)
- successively, creating new population needs more loops: 1) parent selection 2) survival selection 3) recombination 4) mutation

- representation: how to represent elements of A

- the skeleton

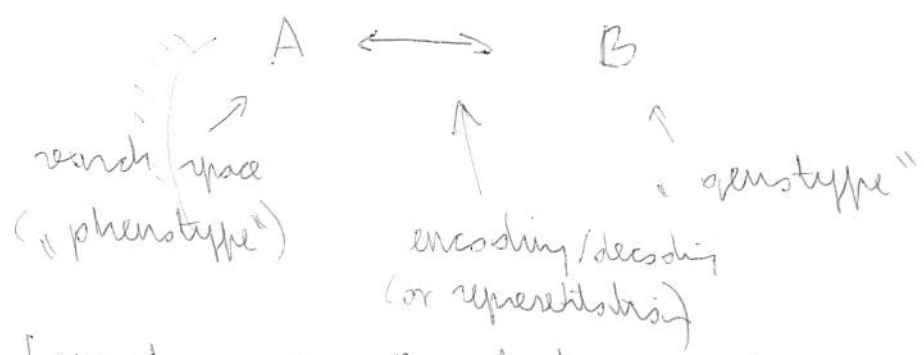
representation
recombination
mutation
parent selection
survival selection

all have variants / implementations that define many flavors of EA-s:

1 initialize population (e.g. randomly)
2 evaluate all elements
3 repeat until termination
4 select parents
5 new solution through recombination of parents
6 mutate new solution
7 evaluate new solution
8 select individuals (from old and new population) for new population
9 end-repeat

evolution strategies, genetic algorithms, genetic programming, even: tabu search, simulated annealing, etc

- some notes (terminology, etc)
- operators:



$x \in A$ (or $x \in B$):
candidate solution or "individual"

$x \in B$:
"chromosome"

[operators use B, but encoding can be the identity mapping (e.g. if $A \subseteq \mathbb{R}^k$) ⇒ one can handle sensible

- unary operation (mutation):

  $m: B \to B$; if $m(a) = a'$: a is „parent", a' is „offspring"

- binary operations

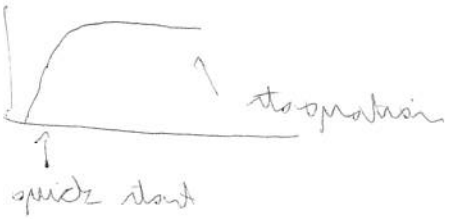
  $b(a, b) = c$, $b: B \times B \to B$ a/b are parents, c is offspring

- termination conditions:

- for example: CPU time, # of evaluations, improvement speed, population diversity

- non-critical; „any time" property. can be stopped at any time, a solution is always available (and improving over time).

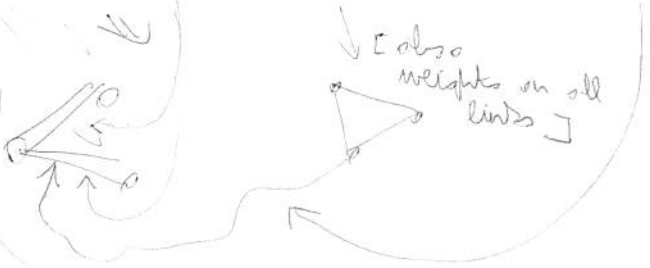
- typical improvement curves:



So, how the videos were made?

- GOLEM project [paper in Nature, 2000 avg]

- representation: <vertices><bars><sensors><actuators> (a numeric vector)



- mutation: for example:
  - change length of bar
  - add/remove vertex
  - attach/detach sensor to bar, etc

- no recombination

- several strategies for selection tested

- fitness: distance a simulated robot covers

[⇒ analytical optimization not a question]

[- robots have no sensors]

[200 element population approx 600 generations initial individuals are empty]

- Karl Sims [now works for Hollywood, and does art installations]
  - representation: 1) a grammar to describe shape, and parameters (eg: depth of recursion, etc) via a dir. graph
    2) all parts have a control system (NN) with sensors, actuators, etc
  - mutation (of directed graphs): add/remove node, etc
  - recombination of directed graphs (specialized operators)
  - fitness: tournaments
  - parent selection: top x% fitness

Evolution Strategies (in 60s by Rechenberg and Schwefel, TU Berlin)

[some schools prefer certain sets of components, depending on application area, taste, history, etc.]

[announcement: 1) www.inf.u-szeged.hu/~jplasty/mi2 2) repeat]
[check memory on skeleton]

- application area

  real valued functions: $A \subseteq \mathbb{R}^n$

  can be discontinuous, noisy, stochastic, dynamic; can have multiple local optima (≠ global optimization); can be "implicit" (black box)

[note: if $f: A \to \mathbb{R}$ is "nice", ES is not the best choice]

- representation: $A \equiv B$ (no encoding)
- recombination: typically none. if yes,:

  a) discrete: $z_i = \frac{x_i + y_i}{2}$

  b) intermediate: $z_i = x_i$ or y_i (chosen randomly)

- parent selection : uniform random
- survivor selection : μ : population size (constant)
  - (μ, λ) ~~= # parents~~ λ : number of parents

  aka « comma » strategy :

  the best μ of λ offspring is the next generation

  λ is typically $\lambda = 7\mu$

- $(\mu + \lambda)$: best μ of $\mu + \lambda$ individuals (old gen. incl.)
- the (μ, λ) strategy is more flexible (adaptive)

  and is recommended (eg : $\mu = 15$, $\lambda = 100$)
- $(1+1)$-ES is stochastic hillclimber

- mutation (most interesting)

$x_i = x_i + N(0, \sigma_i)$ $\quad p_{N(a,\sigma)}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-a)^2}{2\sigma^2}}$

- mutation evolves too (σ) !
- uniform self-adaptive

  individual : $(x_1, \dots, x_n, \sigma)$

  1) mutate σ : $\sigma' = \sigma e^{N(0,\tau)}$ (log-normal change) ← multiplicative

  2) mutate coordinates : $x_i' = x_i + N(0, \sigma')$

  [it is important that we first mutate σ]

  notes : - we require $\sigma' \geq \varepsilon_0$ (to avoid $\sigma \to 0$)

  - $\tau = \frac{1}{\sqrt{n}}$ (n : dimensions) [empirical and some theory too]

  - $N(a,b)$ always means an independent sample (here and in the following)

- heterogeneous self-adaptive

  individual : $(x_1, \dots, x_n, \sigma_1, \dots, \sigma_n)$

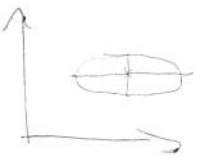
  1) $\sigma_i' = \sigma_i e^{N(0,\tau)} \times e^{N(0,\tau_0)}$ → « global » length mutation

  2) $= \sigma_i e^{N(0,\tau) + N(0,\tau_0)}$

  $x_i' = x_i + N(0, \sigma_i')$

notes : - $\sigma_i' \geq \varepsilon_0$

- $\tau_0 = \frac{1}{\sqrt{2n}}$, $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$



- special case : (1+1)-ES and the 1/5 rule
  - one parameter σ, not mutated
  - but if succes rate (succesful mutations) is low, decrease sigma ; if high, increase.
  - more precisely :
    1) record success rate through $G[\nu n]$ generations
    2) if succes rate $= 1/5$: $\sigma' = \sigma$
       $< 1/5$ $\sigma' = a\sigma$
       $> 1/5$ $\sigma' = \sigma/a$

    where approx $0{,}85 < a < 1$
  - many assumptions : (1+1)-ES, some simple objective functions, one parameter, etc

- correlated mutations

individual : $(x_1, \dots, x_n, \sigma_1, \dots, \sigma_n, \alpha_{12}, \alpha_{13}, \dots, \alpha_{(n-1)n})$

α_{ij} : rotation in plane ij — $n(n-1)/2$ ~~addl~~ parameters

$\Rightarrow$ rotation mx M : - implement n-di rotation with α_{ij} parameters
- multiplication of rotations in pairs of dimensions, like $\begin{pmatrix} \cdots & \cos\alpha_{ij} & -\sin\alpha_{ij} & \cdots & i \\ \cdots & \sin\alpha_{ij} & \cos\alpha_{ij} & \cdots & j \\ & i & j & & \end{pmatrix}$

1) $\sigma_i' = \sigma_i \cdot e^{\tau' N(0,1) + \tau N_i(0,1)}$

2) $\alpha_{ij}' = \alpha_{ij} + N(0,\beta)$ $(\beta \sim 5^\circ)$ $\Rightarrow$ new rotation mx M'

3) $\vec{x}' = \vec{x} + \vec{z}$ where $\vec{z} = M'(N(0,\sigma_1), \dots, N(0,\sigma_n))^T$

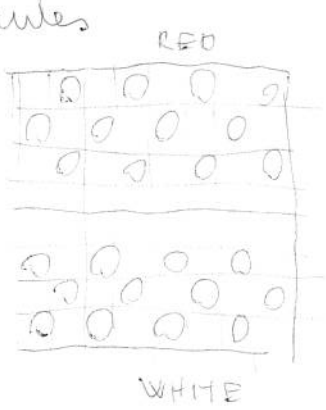
note : in fact, $\vec{x}' = \vec{x} + \vec{N}(\vec{0}, C)$ where C is the covariance mx, and

$c_{ii} = \sigma_i^2$, $c_{ij,ij} = \frac{1}{2}(\sigma_i^2 - \sigma_j^2) \tan 2\alpha_{ij}$

Checkers with Blondie24

- application of ES to teach neural nets to play checkers
- reaches 2045: expert level ranking (unbeaten to date)
- game solved in 2007! result: always stalemate (no legal moves left)
  - CHINOOK program (English tournament, unbeatable since draughts)
  - [Schaeffer et al, Science 317 (5844) pp 1518-1522]

- rules



(chess board)

1) diagonal moves (always on white) one step forward
2) „capture": jump over opponent
3) capture must be made
4) reaching opponents side: become king, can move backwards

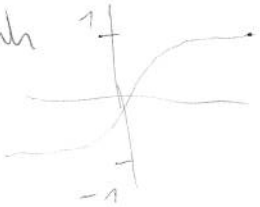
- approach
  - use minimax alpha-beta search, ply depth 4
  - for this, evaluate game states with a neural net
  - teach the neural net with ES, using tournaments

representation

board: 32 element vector (legal positions) where

+K: my king, +1: my regular piece, 0: empty, -1, -K: opponent

standard feed forward neural net with weights on the links, a bias term, and tanh for the nodes as cutoff



- we evolve: weights and bias terms using ES with adaptive σ_i (heterogeneous; we believe)
- we also evolve K (weight of king): $K_i' = K_i + \delta$

δ uniform random $\in [-0.1; 0.1]$

- fitness function: play against 5 random opponents, scores -2, 0, 1 for loss, draw, win.
- selection: (15 + 15) strategy
- 840 generations: the best is picked and put on a website
- conclusions:

  reaches "expert" level (2045 rating)

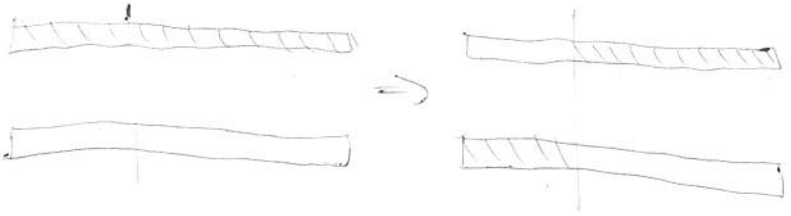
  uses no database or knowledge of any kind but the legal moves

  makes "strange" moves (but good ones) => creativity

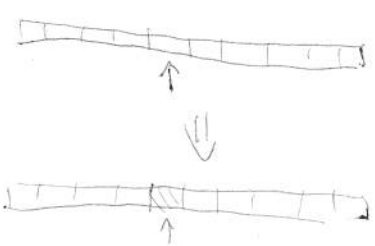
  have been done for chess too: there it reaches master level

Outlook: some other ideas (just briefly mentioned)

- genetic algorithms
  - representation: $B = \{0,1\}^l$ (bit strings)
  - recombination

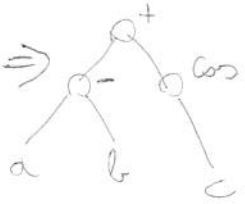


  - mutation



  - parent selection:

    fitness proportional, ranking, tournament, etc

- genetic programming
  - representation: program trees $(a-b)+\cos c \Rightarrow$ 
  - other components: various, intuitive...

- simulated annealing
  same as (1+1)-ES, only we elect the worse solution with prob. t, when t is called "temperature" with $t=0$, we have (1+1)-ES. t is decreased during the run.
- tabu search
  similar to (1+1)-ES, only we generate offspring that has not recently been generated (not on "tabu list").
- direction based methods, like particle swarm, differential evolution (we do not discuss these)
- any hybrid of the above (and any local search technique)! [Most algs. are "hybrid" anyway]
  real-world successful

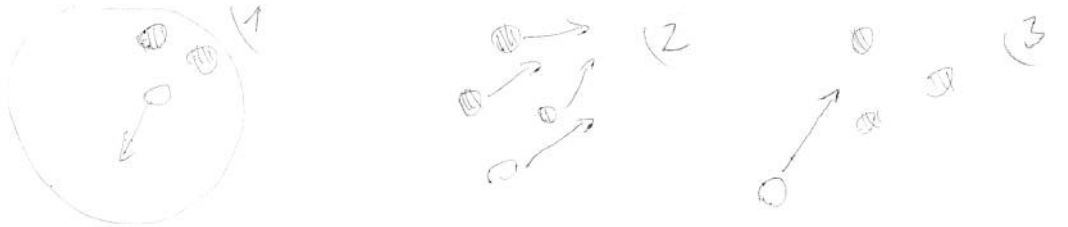
Particle swarm optimization (PSO)

- PSO is part of swarm intelligence
  - ant colony optimization (not discussed)
  [- evolutionary computing too]
  - swarm robotics (self-assembly, formations)
  - P2P computing
  - etc
  $\Rightarrow$ global behavior from local interactions

- Flocking behavior (inspiration for PSO)
  - birds, fish, bacteria, etc
  - basic capacities of individuals, limited intelligence, yet coordinated behavior globally
    - flock and fountain 1)  2)
      (to avoid predators)
    - surround prey (to catch prey)
    - increase efficiency (V shape, etc)

- BOIDS model (Craig Reynolds, '87)
  - we have n particles (birds)
  - a particle i has speed v_i and position x_i (2D or 3D)
  - three simple rules (also for local)
    1 avoidance: move away to avoid collision
    2 copy: follow the direction of others (neighbors)
    3 center: move towards center of flock (neighbors)

⇒ realistic flocking [search for "BOIDS" in youtube]
featured in "Batman Returns"



1) means each particle has "personal space": highest priority rule

$$x_i(t+1) = x_i(t) + \tau\, v_i(t+1) \qquad \tau: \text{step size}$$

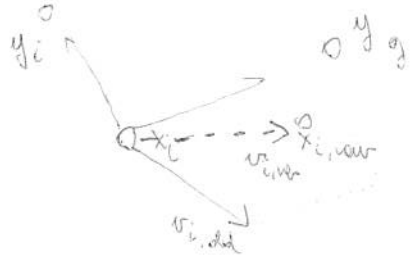
$$v_i(t+1) = \alpha\, v_i(t) + (1-\alpha)\left[\alpha_1 v_{avoid} + \alpha_2 v_{copy} + \alpha_3 v_{center}\right]$$

- v_{avoid} : calculated based on "repel force" of k nearest neighbors
- v_{center} : average x of k nearest neighbors
- v_{align} : average v ——"——

PSO

- application area and purpose: identical to EC (or more ES): global optimization
- PSO is guided by the own experience of birds and information exchange among birds (cognitive and social)
- we have n "particles": $x_1, \dots, x_n \in S$ (usually $S \subseteq \mathbb{R}^d$)
  - a fitness function $f: S \to \mathbb{R}$
  - particle x_i has "velocity" v_i
  - particle x_i has "memory" y_i: the best solution in x_i's trajectory

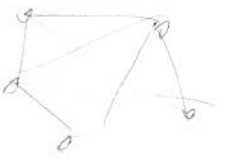
- the algorithm skeleton



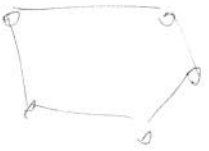
1) initialize particles
2) repeat
3) for $i = 1$ to n
4) if $f(x_i) < f(y_i)$ then $y_i = x_i$
5) $y_g = \min_{j \in Neighbors(i)} y_j$
6) $v_i = v_i + c_1 (y_i - x_i) + c_2 (y_g - x_i)$
7) $x_i = x_i + v_i$
8) end for
9) until not termination

- some notes : three basic components of movement :
  1) previous velocity : "smooth" movement ; and can push particles to unseen regions (v_i)
  2) cognitive component (also "nostalgia") : prevents leaving potentially good regions ($c_1(y_i - x_i)$)
  3) social component : rely on experience of others ($c_2(y_g - x_i)$)

- neighborhood structure
  - "global" : neighborhood is all other nodes : fast convergence, but stuck in local optima



  - "ring" : neighborhood is a ring (also called "local") : global best spreads slower : better in multimodal



  - many experiments with all sorts of neighborhoods

- initialization
  - uniform initialization of particles (x_i) from S
  - $\forall i \quad v_i = 0$ (motion starts by the alg)

- stopping conditions : same as EC : "any time" alg

improvements

- speed clamping : $v_i = (v_{i1} \dots v_{id})$

$$v_{max,j} = \delta \cdot [x_{max,j} , x_{min,j}] \quad (\delta \in (0,1])$$

for a new v_i, all $|v_{ij}|$ are cut off at $v_{max,j}$

- inertia weight :
  - problem with clamping (changes direction, etc)

$$v_i = w\, v_i + c_1(y_i - x_i) + c_2(y_g - x_i)$$

↑ inertia weight

- „cooling" approach

[note simulated annealing, ES, dsl more others have the same problem]

- decrease w / $v_{max,j}$ with time : 1) linearly 2) non-linearly 3) depending on progress, etc...

- fuzzy approach also used (next slides) fuzzy rules:
  e.g. if [normalized best fitness] is LOW, and
  [current w] is LOW then
  [change in weight] is MEDIUM

- parameter settings:
  pop size ~ 30, $c_1, c_2 : \sim U[0,2]$ (uniform sample)
  $w \sim 1$ [- cooling also for c_1, c_2
  - $c_2 >> c_1$: unimodal, large c_1: multimodal]

- relation to previous techniques (EC)
  - same ↑ problem ↑
  - same basic problems: exploration vs exploitation, „cooling", mutation
  - difference: inertia, speed as mutation

- ~~differential evolution~~ [if there is time] [as one addition to previous list of unobscured methods, but after PSO]

[also based on vector differences, but no unbiased speed and there is crossover]

population $x_1, \dots, x_n$

iterate: $v = x_{r_1} + F \cdot [x_{r_2} - x_{r_3}] + \lambda [y_g - x_{r_1}]$

u is new solution $u_j = \begin{cases} x_{r_1,j} & \text{with pr } CR \\ v_j & \text{otherwise} \end{cases}$

if $f(u) < f(x_{r_1})$ u replaces x_{r_1}

[$F \sim 0,8$]
[$CR \sim 0,9$]
[$\lambda \sim 0,9$]

Differential Evolution

- similar to ES or PSO, but fewer parameters

$x_1, \ldots, x_n \in S \; [\subseteq \mathbb{R}^d$ most often$]$

idea: worse like behavior but no concept of speed

instead: random perturbation based on vector differences in population and selection operator (unlike PSO)

- the algorithm:

1) initialize population (random solutions)
2) repeat
3) for $i = 1$ to n
4) $\quad v = x_{r_1} + F \cdot (x_{r_2} - x_{r_3})$ // „mutation"
5) $\quad x_i' = \text{crossover}(v, x_i)$
6) $\quad$ if $(f(x_i') > f(x_i))$ add x_i' to new_population
7) $\quad$ else add x_i to new_population
8) end for
9) population ← new_population
10) until termination

- mutation

r_1, r_2 and r_3 are random samples from the population

where x_{r_1}: vector to be mutated

$x_{r_2} - x_{r_3}$: difference vector

- this is DE/rand/1 variant (mutated vector is random, 1 difference vector)
- other variants: eg DE/best/2

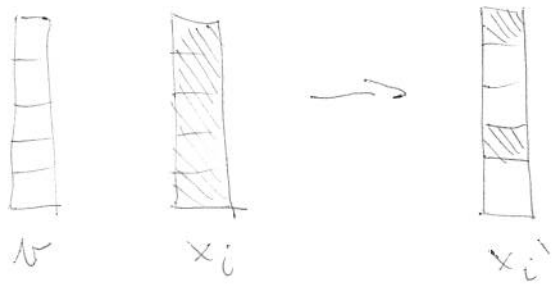
mutated vector is current best solution

there are two difference vectors:

$$v = x_{best} + F \cdot (x_{r_1} - x_{r_2} + x_{r_3} - x_{r_4})$$

this variant introduces more diversity but focuses more on the best solution !

- Crossover



CR : probability of selecting $v_j \to x'_{ij}$ independently for all $j = 1, \dots, n$

$\Rightarrow$ binomial (bin) crossover. eg DE/rand/1/bin

- other variants : exponential (exp) crossover :

1) we sample L from an exponential distr :

$P(L \geq a) = (CR)^{a-1}$ 2) we pick a random position $k \in [1, n]$ 3) we set $(x'_{ik}, \dots, x'_{ik+L}) = (v_k, \dots, v_{k+L})$; the remaining position are copied from x_i. For example, we can have DE/best/2/exp, etc.

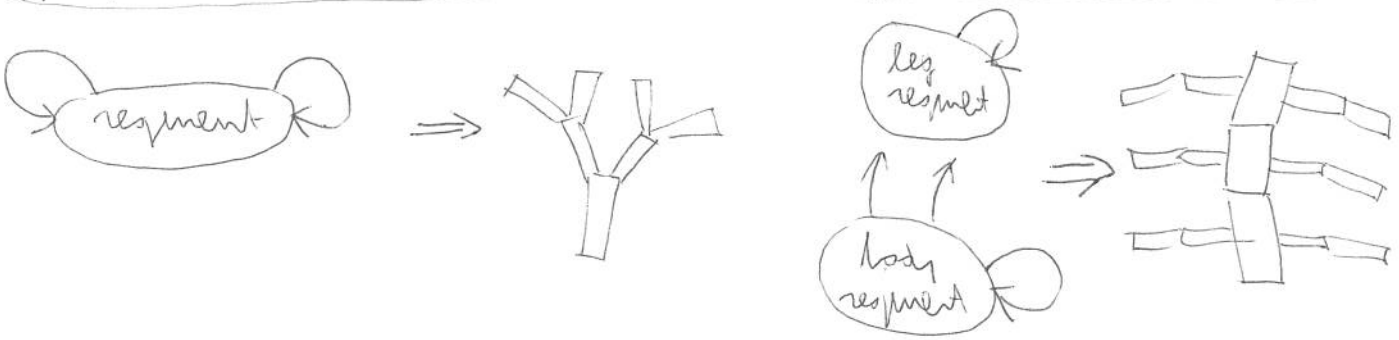
- parameters : $F \sim 0,8$ $CR \sim 0,9$ (if bin) $\sim 0,5$ (if exp)

- point : like advanced variants of ES, respects population shape : if the new elements will be sampled accordingly (mutation vector depends on the population itself, not parameter)

K. SIMS' CREATURES

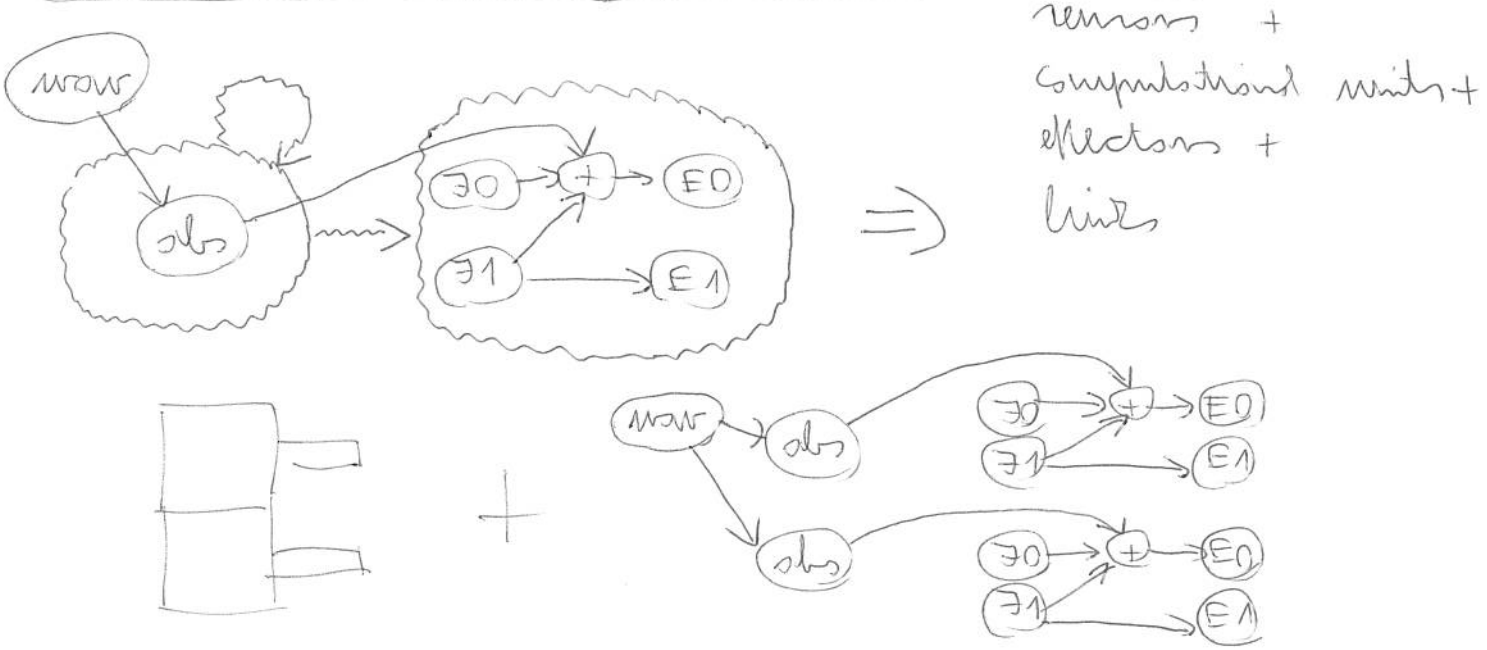
In computer graphics it is difficult to have complexity and control → let's evolve behaviors, so that we have complexity and agent still does what we want (although we do not have explicit control)

— graph language for 3D shapes as representation



- 3D shape is built from a root node, based on attributes:
- each node has attributes: dimensions (shape); joint-type (between the node and its parent) eg rigid, twist, universal, etc; joint-limits (for each degree of freedom); recursive-limit; connection, neurons
- each connection also has attributes
  - placement of child relative to parent: position, orientation, scale, reflection.
  - can be terminal-only (special features such as hand etc)

— the control system and its representation



- sensors: joint angle, contact, photo
- Computing units: sum, product, sin, cos, oscillate, if, etc...
  - each unit has at least one input connection (which can be constant)
  - each connection has a weight, and can be negative or positive
- effectors: one input (with weight), controls one degree of freedom of a joint
- graph language: integrated to morphology language:
  - nodes can have control units that are replicated
  - there are global units
  - connection are between global and node, and parent and child
- <u>various individual fitness</u>
  - 2 locomotion individual creatures are simulated in a virtual world with realistic physics
  - behavior is evaluated (swimming, walking, etc)
  - feasibility checks to remove unfeasible creatures
  - <u>swimming</u>: (1) distance of center of mass covered (2) final speed is weighted more (to reward continuous movement)
  - <u>walking</u>: (1) horizontal speed (2) first simulate with no friction (let it fall over) until center of mass reaches a stable minimum
  - jumping and following similar
- <u>evolution algorithm</u>

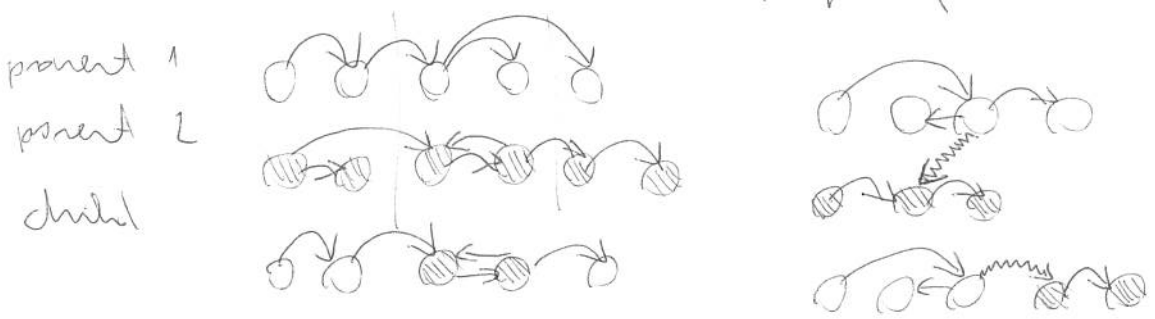
<u>pop size</u>: 300; <u>parent selection</u>: top 20% creatures

<u>survivor selection</u>: parents (60) + new children (240)

- mutation (first choose then conduct mutation)
  - mutation of node attributes: (1) bias mutation with variance proportional to value (2) negation (3) in discrete case picking a new value
  - adding a random node (will need connection too)
  - mutation of connection attributes
  - moving connection point to different node
  - adding new connection, removing existing ones

[lots of parameters! no mention of them in paper]

- recombination
  - crossover
  - grafting



40% asexual, 30% crossover, 30% grafting

- notes: (1) parents are replicated prop. to their fitness
  (2) 50-100 generations, many lines (each line very different subject)

Electric sheep by S. Draves

fractal animations, an example of aesthetic evolution by human selection

- representation: a variant of iterated function system (IFS): an image iteration point that satisfy $S = \bigcup_{i=0}^{n-1} f_i(S)$ where f_i is blend of affine transformations (followed by a "variation", which can be non-linear):

$$f_i(x,y) = \sum_j v_{ij} V_j(a_i x + b_i y + c_i, d_i x + e_i y + f_i)$$

„variations" can be, for example, $V(x,y) = (\sin x, \sin y)$, $V(x,y) = (x/r^2, y/r^2)$, etc. 18 variations are fixed.

Altogether 162 parameters (up to 6 transforms, 24 per transform) define a sheep.

[– summation: the linear parts of the affine transforms are rotated]

– mutation (1) randomizing the v_{ij}-s (2) or the affine transform of one transform (F_i) (3) or adding noise to coefficients. a feasibility check is always produced (too short or long?)

– crossover: (1) swap transforms or (2) pairwise linear interpolation with random factor

– other operators: (1) random rotation (2) user paths

– parent selection: fitness-proportional slowly
steady state: fitness is calculated from user rating

– survivor selection: ~~implicit: population ten~~ ~~opsum~~ remove worst sheep. size of population is approx 30